REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
28-10-2008	Final Report	1-Jan-2008 - 30-Sep-2008

4. TITLE AND SUBTITLE
Towards Trustable Embedded Systems: Hardware Threat Modeling for Integrated Circuits

5a. CONTRACT NUMBER
W911NF-07-1-0648

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
611102

6. AUTHORS
Jia Di

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES
University of Arkansas
Board of Trustees
University of Arkansas
Fayetteville, AR 72701 -

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
53318-CI-II.1

12. DISTRIBUTION AVAILIBILITY STATEMENT
Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT
As integrated circuits become more complex, it becomes easier to hide malicious logic constructs within a design. Security-conscious hardware designers require a way to detect such logic embedded in Third Party IP blocks used by their designs. The aim of this project was to develop a systematic way to detect attacks implemented in a design. Using the Java programming language, a tool capable of producing an attacker-centric threat model was developed. The tool uses a library of predefined malicious patterns to detect and categorize attacks in a system. Upon completion, the tool was tested on a small RISC microprocessor containing denial of service and data tampering attacks. Once given the appropriate library pattern, the

15. SUBJECT TERMS
embedded system security

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Jia Di
U	U	U	SAR		19b. TELEPHONE NUMBER 479-575-5728

Report Title

Towards Trustable Embedded Systems: Hardware Threat Modeling for Integrated Circuits

ABSTRACT

As integrated circuits become more complex, it becomes easier to hide malicious logic constructs within a design. Security-conscious hardware designers require a way to detect such logic embedded in Third Party IP blocks used by their designs. The aim of this project was to develop a systematic way to detect attacks implemented in a design. Using the Java programming language, a tool capable of producing an attacker-centric threat model was developed. The tool uses a library of predefined malicious patterns to detect and categorize attacks in a system. Upon completion, the tool was tested on a small RISC microprocessor containing denial of service and data tampering attacks. Once given the appropriate library pattern, the tool was able to detect both threats in the design.

List of papers submitted or published that acknowledge ARO support during this reporting period. List the papers, including journal references, in the following categories:

(a) Papers published in peer-reviewed journals (N/A for none)

Number of Papers published in peer-reviewed journals: 0.00

(b) Papers published in non-peer-reviewed journals or in conference proceedings (N/A for none)

Number of Papers published in non peer-reviewed journals: 0.00

(c) Presentations

Number of Presentations: 0.00

Non Peer-Reviewed Conference Proceeding publications (other than abstracts):

Number of Non Peer-Reviewed Conference Proceeding publications (other than abstracts): 0

Peer-Reviewed Conference Proceeding publications (other than abstracts):

Number of Peer-Reviewed Conference Proceeding publications (other than abstracts): 0

(d) Manuscripts

Number of Manuscripts: 0.00

Number of Inventions:

Graduate Students

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Names of Post Doctorates

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Names of Faculty Supported

NAME	PERCENT_SUPPORTED	National Academy Member
Jia Di	0.16	No
FTE Equivalent:	**0.16**	
Total Number:	**1**	

Names of Under Graduate students supported

NAME	PERCENT_SUPPORTED
Michael Hinds	0.50
Jordan Yust	0.50
Wiwat Leebhaisomboon	0.50
Michael Linder	0.50
Chris Porter	0.50
Jeremy Choens	0.50
FTE Equivalent:	**3.00**
Total Number:	**6**

Student Metrics

This section only applies to graduating undergraduates supported by this agreement in this reporting period

The number of undergraduates funded by this agreement who graduated during this period:	5.00
The number of undergraduates funded by this agreement who graduated during this period with a degree in science, mathematics, engineering, or technology fields:.....	5.00
The number of undergraduates funded by your agreement who graduated during this period and will continue to pursue a graduate or Ph.D. degree in science, mathematics, engineering, or technology fields:.....	5.00
Number of graduating undergraduates who achieved a 3.5 GPA to 4.0 (4.0 max scale):	4.00
Number of graduating undergraduates funded by a DoD funded Center of Excellence grant for Education, Research and Engineering:	0.00
The number of undergraduates funded by your agreement who graduated during this period and intend to work for the Department of Defense	4.00
The number of undergraduates funded by your agreement who graduated during this period and will receive scholarships or fellowships for further studies in science, mathematics, engineering or technology fields:	0.00

Names of Personnel receiving masters degrees

NAME
Total Number:

Names of personnel receiving PHDs

NAME
Total Number:

Names of other research staff

NAME	PERCENT_SUPPORTED
FTE Equivalent:	
Total Number:	

Sub Contractors (DD882)

Inventions (DD882)

The Trustable Recognition of Undesired Threats in Hardware (TRUTH) Analysis Tool – An Approach to Hardware Threat Modeling

D.1 Introduction

The impact of software viruses has been felt by the entire computerized world, through loss of productivity, loss of system resources or data, or mere inconvenience on a massive scale. Hardware, especially integrated circuits (ICs), was considered safe and attack-free, in contrast to its software counterpart. However, as technologies advance and markets expand, hardware is becoming vulnerable like software. Malicious logic, similar to a software virus, could be inserted into a circuit like a Trojan horse, such that it lies dormant and is very difficult to detect until activated, but then cannot be effectively defeated. These days most complex digital systems are not designed from scratch; instead they use many Third Party Intellectual Property (IP) blocks. Hence, one or more Third Party IP blocks could contain malicious logic that may affect the entire system. The inserted malicious logic can lead to various unwanted scenarios that threaten the three key aspects of information security: Availability (Denial-of-Service), Confidentiality (Information Leakage), and Integrity (Data Tampering). These hardware threats must be identified and the corresponding attacks must be mitigated to ensure that the IC is trustable, i.e., only performs functions defined in the original circuit specification (no more and no less), before applications employing these chips are placed into operation. Note that the aims of malicious logic attackers are beyond profit-seeking; they also include issues that may affect national security.

D.1.1 Approach

Threat modeling is a method of assessing and documenting the security risks associated with an application. A fundamental component of any formal analysis of security properties, including those for hardware, is to construct an appropriate threat model for the environment in which the system is designed to be working. It is essential to ensure the threat model is neither too weak nor too strong: if too weak, the system may pass the check but may not be safe in field operation; if too strong, unnecessary design constraints may be posed to the designers.

The key to effective threat modeling is to efficiently and systematically check for inserted malicious logic in an IC. Once an effective method has been implemented, one can proceed in developing a more trustable IC. The malicious-logic checking mechanisms needed for such a method can be classified into two categories: attacker-centric and defender-centric. For this project, the Trustable Recognition of Undesired Threats in Hardware (TRUTH) Analysis Tool utilized attacker-centric checking mechanisms in its approach towards threat modeling.

Software-based attacker-centric checking mechanisms are separate from the design effort, and search the entire circuit structure after the design is complete. Therefore, no additional circuitry is added to the original design. The attacker-centric checking mechanism used in the TRUTH Analysis Tool is a variation of the mechanism known as Structural Checking [1]. Structural Checking focuses on circuit structure as opposed to functionality – it searches the IC circuit structure for potential malicious logic

after the design is complete. It is most effective for the following types of malicious logic:

- *Malicious logic targeting signals having multiple sources and destinations* – these signals are also called "traceable" signals, whose sources and destinations can be easily traced by inspecting the IC's bi-directional linked-list. Examples of such signals include clock tree, busses, RESET signal, interrupt signal, primary inputs and outputs;
- *Malicious logic targeting combinational circuits with a reasonable number of I/O ports* – the malicious logic inside a combinational circuit is activated by the circuit inputs and propagates its effect to the outputs. Therefore, Structural Checking can either start from the output lines and "trace back" into the circuit to check the circuit blocks controlling these outputs, or from the input lines to search for the circuit blocks being controlled by these inputs. To reduce complexity, the number of circuit inputs/outputs cannot be too many;
- *Malicious logic targeting small combinational circuits* – for small combinational circuits, even if they have many input/output ports, Structural Checking is still applicable since the total circuit complexity is low.

The TRUTH Analysis Tool utilizes the Structural Checking attacker-centric checking mechanism in its approach to threat modeling. An in-depth analysis of how the Tool uses this checking mechanism is presented in the sections that follow.

The Structural Checking approach is applicable to all levels of abstraction, from RTL description to physical layout. However, application to transistor-level or physical-level netlists is more difficult than RTL or gate-level netlists, because of greater difficulty in identifying logic constructs at these lower levels of abstraction. For this reason, the TRUTH Analysis Tool performs threat modeling only at the RTL level.

D.2 Functionality

D.2.1 Overview

The objective of the TRUTH Analysis Tool is to perform threat modeling on an IC design accurately and efficiently at the RTL level of abstraction. Given the design layout and information provided by the user, the Tool analyzes the circuit design for likely attacks. It then provides a list of these attacks to the user to allow for further inspection of the locations in the circuit where the attacks occur. When in the process of analyzing a circuit for attacks (so-named "risk analysis"), there is a tradeoff between the thoroughness of the analysis and the resources needed to conduct such an analysis. A more thorough analysis can only be conducted at the expense of computation time and resources. For this reason, the Tool attempts to optimize the tradeoff and maximize the effectiveness of the resources used.

D.2.2 Tool Structure

The TRUTH Analysis Tool takes a step-by-step approach to performing threat modeling on an IC. Throughout each step, the Tool acquires the attributes necessary to effectively analyze the IC for possible attacks.

The Threat Model, denoted as M, of a target IC, can be defined as $M = \{S, P, C, A, T, K\}$. S is the internal circuit structure of the target IC, and P represents

the intended application. $c_i \in C, i \in [1,n]$ represents each Third Party Intellectual Property (IP) block, where n is the total number of Third Party IP blocks, and $a_j \in A, j \in [1,m]$ is each asset of the target IC, where m is the total number of assets. T is the library of threat patterns used to analyze the circuit and K is the set of corresponding attacks that result after conducting risk analysis.

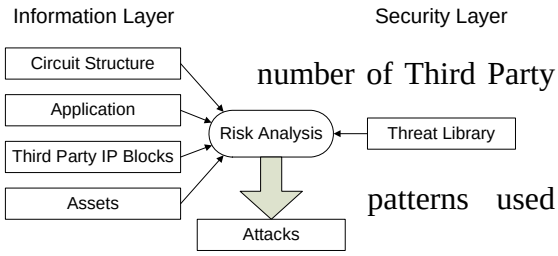

Figure 1: TRUTH Analysis Tool

D.2.3 Creating the Circuit Structure

As stated, the TRUTH Analysis Tool accounts for circuits represented by an RTL description, which is the most abstract level of design. This type of description is written in VHDL (VHSIC Hardware Description Language). The VHDL parser written by Dr. Christoph Grim, et. al., provides a framework that translates the IEEE 1076 standard (Design Automation Standards Committee of the IEEE Computer Society) into Java code [2]. Using object-oriented programming in Java, the parser has been highly modified in order to build the internal circuit structure S needed for threat modeling. Upon reading in a VHDL file, the parser creates a representation of the file in memory. It then recognizes specific attributes about the file that allows certain automated operations necessary for risk analysis to be performed. These operations include tracing data paths, recognizing functional blocks and memory elements, and determining points at which signals are modified.

The attributes recognized by the parser are shown in Figure 2 and are the following:

- *Signals* - the term “signal”, in this context, represents a connection between two or more digital components. Each signal is either a port signal or an intra-signal. Port signals (also known as I/O signals) connect from outside the circuit to one or more of the circuit's subcomponents. Each port signal has a “mode” characteristic that represents how data flows through that signal (e.g., IN, OUT, etc.). Intra-signals, on the other hand, represent the circuit's internal wires connecting two or more subcomponents together. All intra-signals have a “mode” characteristic of SIGNAL, meaning their data flow does not have a defined direction.
- *Process blocks* – a process block is a logic block typically used to build sequential circuits. These circuits act only when certain signals change their value, which are listed in the process block's sensitivity list. The parser treats a process block as a subcomponent where signals accessed by that process (in the sensitivity list or inside the block) are connections to the rest of the circuit. In other words, the parser determines which of the circuit's port and intra-signals are accessed from a process block and then generates a list of those signals for that process block.

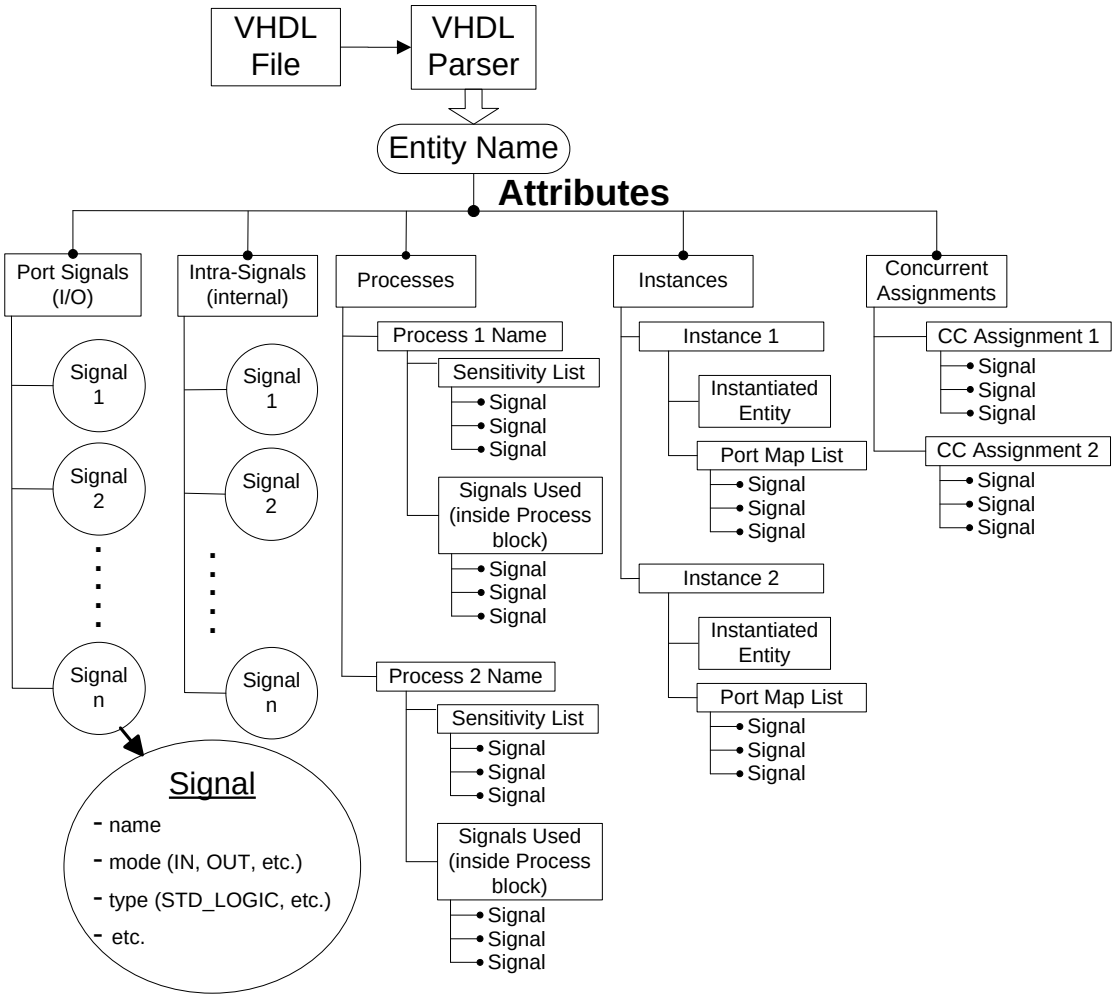

Figure 2: Attributes of a circuit recognized by the VHDL parser.

- *Concurrent assignments* – concurrent assignments are logical operations performed simultaneously and continuously inside a circuit as opposed to sequentially. These assignments are defined inside the circuit's architecture and outside process blocks. The parser represents each concurrent assignment as a sequence of signals, operations, and literals (e.g., '1', "000", X"ABCD0123", etc.) verbatim as it appears in the statement.

- *Subcomponents (Instances)* – when a VHDL file is parsed, its subcomponents are instantiated as instances and the VHDL files corresponding to those instances are placed into a queue to be parsed next. This is necessary in order for the circuit structure *S* to properly represent the actual circuit design.

Most circuit designs are implemented in a hierarchy of sub-circuit designs, each of which is coded in a separate VHDL file. In order to preserve this hierarchy, a depth first approach is used to create a logical representation of the design in memory. To begin creating the circuit structure, the top-level design file is read in by the parser. As the file is being parsed, its subcomponents are instantiated as instances and the VHDL files corresponding to those instances are then parsed. This recursion continues until the parser reaches a file that has no subcomponents. After each file of the design has been

parsed, connections between each file (signals) are organized so that data paths can be followed. The structure mirrors that of the actual VHDL hierarchy representation used to create the circuit design.

D.2.4 Specifying the Application

ICs can have multiple intended applications at the same time, ranging from aircraft control to data encryption. For each application of an IC, an attacker's goals are likely to change. Therefore, when identifying threats, one needs to ask: what is the *primary* use of this IC? The Tool provides a list of applications separated into categories to choose from. The user is to select the application that best describes the primary use of the target IC. Included in each application is its own set of "threat levels". Since the ease and severity of accomplishing the three types of threats (Denial-of-Service, Information Leakage, and Data Tampering) is dependent on the nature of the application, certain threats are easier to implement and cause more damage in some applications compared to others. For this reason, each application is given a rating (threat level) for each threat. The rating ranges from 0 to 10, with 10 being the most severe, and combines the likelihood of implementing the threat with the severity it would cause into an integer value. For example, an IC associated with handling the results of a voting machine would have a very high Data Tampering threat level due to the repercussions that would ensue if the machine were successfully compromised. The threat levels from the chosen application are then taken into account when conducting risk analysis.

D.2.5 Determining Third Party IP blocks

Most complex ICs today are not developed solely by the user. Instead, most ICs include 3rd party Intellectual Property (IP) blocks in their implementation. Since these IP blocks did not originate from the user, one or more could contain malicious logic that may affect the entire system. Therefore, it is important to distinguish between blocks in the circuit design that are trustable as opposed to those that are not trustable.

The TRUTH Analysis Tool requires this specification. Upon selecting an appropriate application for the IC, the Tool requires the user to specify any and all Third party IP blocks included in the design. Since the Tool performs threat modeling on ICs at the RTL level written in VHDL only, it considers "Third Party IP blocks" to be any VHDL files of the design not developed by the user nor any other party completely trusted by the user. In other words, a Third Party IP block is any file originating from individuals not directly affiliated with the user or company for which the TRUTH Analysis Tool is performing threat modeling. This is because the likelihood of malicious logic contained in one of the VHDL files, however great it may be, is still a *realistic possibility* and should be handled as such. In addition, VHDL files originating from the user or a trusted party are considered completely trustable with no possibility that malicious logic is contained in the files. These files are not taken into account when performing risk analysis. Therefore, specifying which files are Third Party IP blocks allows the Tool to refine its scope to a more specific range of IC components when performing threat modeling.

Similar to the concept of having different "threat levels" for different applications, the task of specifying Third Party IP blocks allows for variations to occur. How trustable a Third Party IP block is varies for each block. Some blocks are (and should be) considered more trustable than others. For this reason, each Third Party IP block has a

"trustability weight" associated with it. This weight is determined by the user and is aimed towards providing a more accurate description of the tagged Third Party IP block. The weight ranges from 0 to 10, with 0 being completely trustable (all VHDL files not marked as Third Party are automatically given a weight of 0). The determined trustability weight is then taken into account when performing risk analysis. For this reason, it is important to remain consistent throughout the process of assigning weights to Third Party IP blocks. Some suggested questions to help maintain consistency when assigning trustability weights are as follows:

- *What is the reputation of the company from which this VHDL file originated?*
- *Has there been a history between the user and this company in the past?*
- *If so, was their work sufficient?*
- *Have any other companies had privacy issues occur when conducting business with this company?*

It is important to note that these questions are only suggestions to aid the user. They are aimed towards promoting consistency throughout the process of assigning trustability weights to Third Party IP blocks. Given an actual situation, one should develop a unique set of guidelines that is relevant to the target IC and its intended use.

D.2.6 Specifying Assets

The process of assigning assets aims to answer the question "why would an attacker attack this IC?" This is critical because the ultimate purpose of threat modeling is to prevent attackers from fulfilling their goals. In order to achieve this, the attackers' goals must be understood. An asset is an abstract or concrete resource that a system must protect from misuse by an adversary [3]. It is impossible to have a threat without a corresponding asset(s) because assets are essentially threat targets. They are used to describe circuit components based on the components' contribution to the overall functionality of the IC. For this reason, components are usually assigned multiple assets in order to fully describe their use and contribution.

The TRUTH Analysis Tool relies highly upon user interaction throughout the process of assigning assets. This process is by far the most time consuming and difficult to comprehend due to its counterintuitive nature, but is vital to the effectiveness of the Analysis Tool. An inconsistent or inaccurate set of assets leads to an improper set of attacks to be determined during risk analysis which ultimately renders the Tool as ineffective. The requirements for assigning assets are rather extensive and are discussed in detail in the sections that follow. However, it is important to keep in mind that the process of selecting assets provides necessary information and is essentially the backbone of the TRUTH Analysis Tool.

D.2.6.1 Asset Descriptions

As stated, assets are used to describe circuit components based on the component's contribution to the overall functionality of the IC. In other words, assets are the means an attacker can use to achieve one of the three possible threats: denial-of-service, data tampering, or information leakage. Naturally then, there are several different kinds of assets needed to fully describe a circuit's components. Shown below are all the possible assets the user can choose from when describing a component's

contribution.

- *Control* – category of assets related to the control and operation of the circuit
  - *DataSensitive* – affecting the control of data flow throughout the circuit (ex. the system bus)
  - *OperationSensitive* – affecting the overall operation of the circuit (ex. enable signals)
  - *TimeSensitive* – describes components that affect the timing of a circuit (ex. clock signals)
- *Data* – category of assets related to the transfer and manipulation of data
  - *Memory* – affecting memory data transfer and storage (ex. general registers)
  - *Computational* – affecting data generation and interpretation (ex. ALU, Data Mux)
  - *Critical* – affecting important data crucial to the circuit (ex. passwords, encryption keys, etc.)
- *LogicBlock* – category of assets only available for concurrent assignments, instances, and processes. This category is based more on a component's structure and origin as opposed to its use. It is defined as any circuitry containing logical gates.
  - *FSM* – logic block behaving like a finite state machine
  - *Combinational* – category of logic blocks that operate concurrently
    - *ALU* – combinational logic block behaving like an ALU
    - *DECODER* – combinational logic block behaving like a decoder
    - *ENCODER* – combinational logic block behaving like an encoder
  - *Memory* – logic block behaving like a memory component
  - *Other* – logic block that is not represented by any other asset in the "LogicBlock" category
  - *ThirdPartyIP* – logic block that was not developed by the user (i.e., it came from a Third Party vendor)
  - *Tristate* – logic block behaving like a tri-state buffer
  - *Sequential* – logic block operating sequentially (i.e., process blocks)

It is important to note that the *categories* of assets are also shown. However, these categories serve only for organizational purposes and are not among the options the user can select from.

D.2.6.2 Asset Specification Requirements

The process of assigning assets is completely dependent on the set of Third Party IP blocks, *C*, which was explained in a previous section. The contents of *C* determine which components of a circuit need to have assets assigned. For this reason, the TRUTH Analysis Tool requires the set *C* to be fully developed prior to assigning assets. Each VHDL file included in *C* resides in its own location in the circuit structure *S*. Since the circuit structure is represented in a hierarchy-type fashion, each file in set *C* is located on a different "level" in the circuit design – the farther into the circuit hierarchy, the higher the level. For example, the top-level design file is always located on level 0, and any file instantiated in the top-level file is located on level 1. Therefore, since each file in set *C* is located at a specific level, the Tool is able to determine the *deepest* level that a VHDL

file from set *C* is located. In other words, given a set of Third Party blocks determined by the user, the Tool is able to find the farthest level into the circuit that one of those blocks is located. It does this by simply looping through each file in set *C* and determining which file is located the farthest into the circuit. That level is labeled as the deepest Third Party level throughout the remainder of the TRUTH Analysis Tool. In the situation where a VHDL file is instantiated several times throughout a design, the Tool considers the level of the file's farthest instantiation to be the file's level. Therefore, if a file is instantiated at level 1 as well as level 2 in a design, then the Tool considers the file's level to be level 2 and will handle it as such.

Once the deepest Third Party level has been determined, the Tool requires the user to specify assets for all components down to that level, but not for the level itself. This means that all logic blocks (processes, instances, and concurrent assignments) and signals located before this level need to be assigned the appropriate assets before risk analysis can be performed. For example, suppose the top-level VHDL file ("file_A.vhd") instantiates another VHDL file ("file_B.vhd") which then instantiates yet a third VHDL file ("file_C.vhd"). Therefore, the way the circuit structure *S* would appear is as follows:

- Level 0 → file_A.vhd : top-level
- Level 1 → file_B.vhd
- Level 2 → file_C.vhd

If, when selecting Third Party IP blocks, the user determined that "file_B.vhd" was Third Party and no other file, then the user would only be required to specify assets for all components located before level 1. If, in the same example, the user determined that "file_C.vhd" was Third Party, then the deepest Third Party level would be level 2 and the user would be required to specify assets for all components in level 0 as well as level 1.

Furthermore, tagging assets is on an Instance-by-Instance basis. VHDL files can be instantiated many times throughout a design and, therefore, can represent different logical components of the design with each instantiation. Each of these instantiations could handle different types of data based on where in the design it is instantiated which then would require different assets for each instantiation. For example, a VHDL file representing a tri-state buffer could be instantiated numerous times through a design. Each one of those instantiations could represent a different logical component and, therefore, be handling different types of data. Since the data would be different with each instantiation, it would contain different assets. For this reason, the user is required to specify assets for components at every *instantiation* of a VHDL file as opposed to the VHDL file itself.

In many situations, designs contain several Third Party IP blocks. These files could be (and usually are) located on different levels of the circuit. As explained above, the user is required to specify assets for all components down to the deepest Third Party level, but not for the level itself. However, in the case that a Third Party VHDL file is located on a level that isn't the deepest level, the user is not required to specify assets for all components but rather only for the I/O signals of that file. This is because, since the file came from a Third Party vendor, the user did not create the file. Therefore, the user will not have enough knowledge about the components of the file (e.g. internal signals,

process blocks, etc.) to accurately determine the appropriate assets for each component. The only components of the file the user can accurately and confidently assign assets to are the I/O signals of the file. For this reason, the user is only required to specify assets for those signals and no other component.

D.2.6.3 Asset Filtering

Since most complex IC's today are not developed solely by the user, it is likely the user may not know all the assets of every component in the IC. However, assets provide important characteristics to each component in the circuit design and are heavily relied upon when performing risk analysis. Therefore, it is crucial to have a complete and accurate set of assets for each component of the design.

Once the user has specified all the necessary assets for a design, a process known as *asset filtering* takes place. This process automatically filters assets through signal components that are related in order to maximize the accuracy of each signal's set of tagged assets. The process, which is shown in Figure 3, filters assets in a depth first fashion. It begins with the top-level design file and filters assets in two different ways: vertically through the design hierarchy (Instance Filtering) and laterally through signal connections (Concurrent Assignment Filtering).

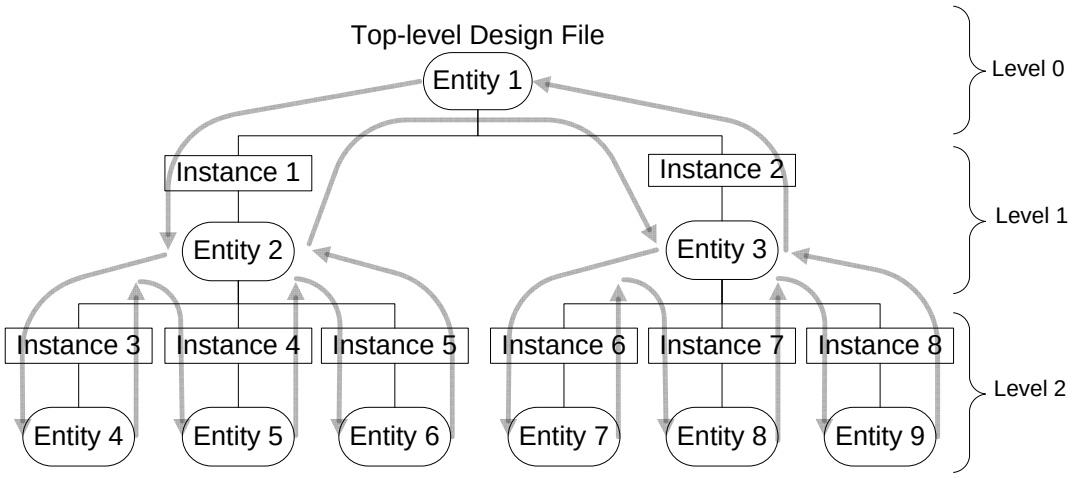

Figure 3: The process of filtering assets through a VHDL design hierarchy.

D.2.6.3.1 Instance Filtering

Filtering assets through instances is highly beneficial when inconsistent sets of assets are tagged to signals connected via an instance. This case arises when a signal included in the instance declaration contains different assets than the corresponding lower level signal in the instance's design file. When this occurs, each signal's assets are *filtered* across the instance and applied to the other design file's corresponding signal. The end result, therefore, is that both signals are tagged with the same set of assets. This ensures there are no inconsistencies in assets when traversing through the instances in the circuit.

D.2.6.3.2 Concurrent Assignment Filtering

Filtering assets through concurrent assignments is a way to distribute user-tagged

assets from one signal to another. Similar to instance filtering, its goal is to maximize the accuracy of signal assets before performing risk analysis. There are two types of concurrent assignment filtering that take place, and both operate in a bit-wise or vector-wise fashion, depending on the type of operators inside the concurrent assignment statement. If the concurrent assignment contains at least one mathematical operator, then the filtering will be in a vector-wise fashion. However, if it only contains logical operators, then the filtering will be in a bit-wise fashion, where all bits in the vector are independent of each other. Both types of filtering processes are explained below.

- *Driven signals* ➔ *Driving signals*

  In this filtering process, assets are transferred from signals being driven to their driving signal counterparts. The only time this filtering takes place is in a situation where a signal is being driven by itself and some other signal(s), which is shown in Figure 4.

  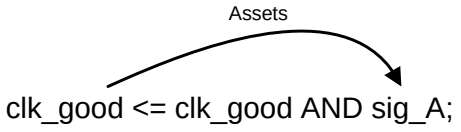

  Figure 4: Concurrent Assignment Filtering – driven signals to driving signals

  The signal *clk_good*, which represents the system clock, most likely contains an accurate set of assets defined by the user. However, if *sig_A* is defined internally within the VHDL file, the user might not be able to accurately define a complete set of assets for it. Since *sig_A* directly affects *clk_good* (if *sig_A* is logic '0', then the clock can essentially be shut down), it needs to have all assets assigned to *clk_good*. Therefore, using this filtering process, *sig_A* will be assigned the assets currently assigned to *clk_good*, thus ensuring the accuracy of the signal's assets.

- *Driving signals* ➔ *Driven signals*

  The other type of concurrent assignment filtering is the one that primarily takes place in concurrent assignments. Assets from the driving signals are filtered to the corresponding signals of which they are driving. This type of filtering usually is performed in a bit-wise fashion, meaning assets of driving signals are filtered to the *exact* signal they are driving, and no other signal.

  

  Figure 5: Concurrent Assignment Filtering – driving signals to driven signals

  Shown in Figure 5 is an example where this filtering would take place. Vector *A* represents a 4-bit vector being driven by two other 4-bit vectors, *B* and *C*. Since both *B* and *C* directly determine the values of *A*, their assets need to be assigned to *A*. Therefore, when filtering takes place, all assets from *B* and *C* will be filtered to *A*. Furthermore, since the assets are filtered in a bit-wise fashion, each asset from *B(0)* and *C(0)* will be filtered to only *A(0)*, *B(1)* and *C(1)* will be filtered to only *A(1)*, and so on. This ensures no assets are incorrectly assigned to signals.

D.2.7 Risk Analysis

Risk analysis is at the center of the tool's ability to recognize and categorize potentially malicious logic. Its accuracy, however, is dependent in large part on properly assigning assets to components of the circuit structure *S*. These assignments include both the user-interactive assignment process as well as asset filtering. Risk analysis operates on one VHDL file at a time. Therefore, for a multilevel circuit, this process may occur many times. Using this type of method allows the process to be modular, and the depth and computational requirements of risk analysis can be modified for individual needs.

There are three main components that comprise risk analysis: path trees, path analysis, and risk assessment. Each of these components is explained in detail in the sections that follow, but the general order of operations of the risk analysis process contains four steps. To begin, it builds a set of path trees from the circuit structure *S*. Then, the path analyzer searches each path tree for malicious asset patterns by comparing them with those contained in the Threat Library. The output from the path analyzer, which is a list of potential threats, is then sent to the trustability calculator to perform risk assessment. Finally, after risk assessment, a list of potential attacks, along with information about the attacks, are displayed to the user.

D.2.7.1 Threat Library

The Threat Library contains a compilation of all known malicious asset patterns. This Library is designed to be expandable so that, as more knowledge accrues of how malicious code is implemented in hardware, the Library will be able to accept more precise and reliable asset patterns. This implies that the Library should be easily editable by the user. At present, an automated feature for adding asset patterns is not available in the TRUTH Analysis Tool; however, the user has the freedom to manually update the Library with a new asset pattern by editing the Library's text file, ThreatLibrary.lib. It is crucial to preserve alphabetical order when editing the Library file because failure to do so may lead to inaccurate results.

In the Library, potential threats are stored as a sorted list of malicious asset patterns. In addition, the Library contains information such as the type of attack (Denial-of-Service, Data Tampering, or Information Leakage) and a number giving the reliability of each asset pattern. The more uniquely the pattern matches an actual threat, the higher the reliability rating. Once the Library is ready to be used, it is instantiated into a tree structure, which is shown in Figure 6. Since the Library is sorted, converting it into a search tree is simple and efficient. First, the Library text file is read line by line, where each line contains a malicious asset pattern and corresponding data. The pattern is broken down into individual assets that are inserted into a tree structure, as shown in the Figure. Since the tree is sorted, it is likely that several lines will contain the same leftmost assets, i.e., A3 and A2. These form the root of the tree. Progressing to the right, the tree is traversed along the matching path of assets from the current line of the

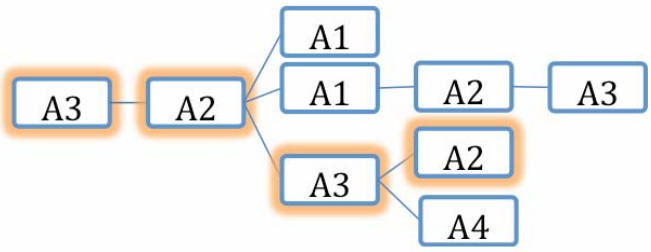

Figure 6: Library Asset Tree with an asset pattern highlighted.

Library's text file until the line of the text file differs from the tree. When this occurs, a new branch is added to the tree that stretches to the end of the line. The end result is a tree structure similar to the one shown in the Figure.

Highlighted in the diagram is a potentially malicious logic pattern. Each pattern remains distinct within the tree through its leaf node, which in this case is asset A2. Therefore, there is a one-to-one correspondence between asset patterns and leaf nodes. Each leaf node in the parsed tree contains the asset pattern's type and reliability number. Once the Library text file has been parsed it is ready to be used in path analysis.

D.2.7.2 Path Analysis

Path analysis is a similar process to the malicious code detection of software anti-virus programs. It begins by manipulating the circuit structure into a searchable graph, called a path tree, as shown in Figure 6. Next, each path in the graph is divided into a linear list, called a working list, and the working list's asset patterns are compared to those contained in the Threat Library. As matches are found, they are sent to a queue to be assessed, consolidated, and displayed.

Building a path tree from the parsed circuit begins by identifying all primary inputs and outputs of each VHDL file. Once these are identified, the method begins at a primary output of the file, identifies all logic blocks driving the output, and recursively navigates to each. As the method visits each logic block, it identifies the block's input signals, recursively visits them, and then visits their respective driving logic blocks. This recursive process continues until a primary input is reached. Once a primary input signal is encountered, it is made a leaf node and the recursion bottoms out. Special care must be taken when a circuit contains feedback loops in order to avoid infinite looping. Feedback loops occur when a logic block connects to a signal that connects to another logic block, which connects to another signal and so on until the original logic block is reached. A "loop detector" checks for cycles in the path trees and handles feedback loops in the circuit. It locates the beginning of the loop and returns one level in the recursion to the next signal outside the loop, thus linking the loop back to its starting logic block. After all paths from the primary output have been combined into a tree, the method moves on to the next primary output. Finally, after all primary outputs obtain their respective path tree, the trees are combined into an array and sent to the search process.

The search process iterates through each path tree and creates a working list for each path in the tree. The working list, shown in Figure 7, contains the asset patterns that are to be compared with the Threat Library. As each working list is created, its assets are linearly compared with the patterns in the Library. Since the threat patterns in the Library allow for multiple assets to be contained within the same signal or logic block, a multi-way comparison algorithm is used to determine matches.

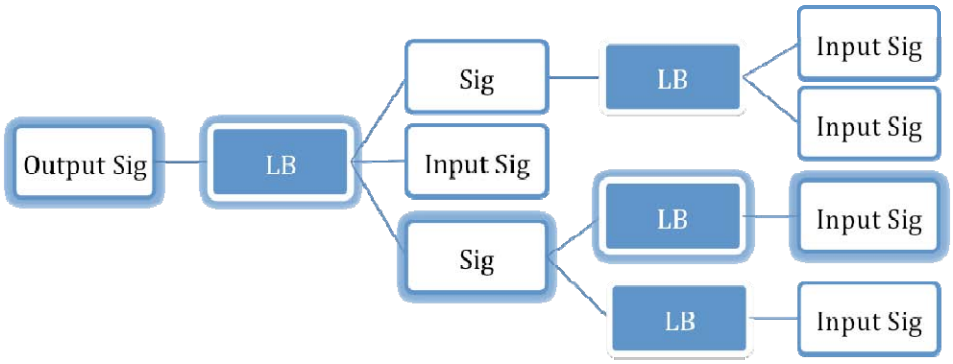

Figure 7: Path tree with a working list highlighted.

If a full match is achieved, an attack data structure is created that contains the logic blocks, the asset pattern, and the library information of the malicious pattern. These attack structures are passed to the risk assessment method to be assigned a trustability rating. Due to overlapping of the path trees, a single malicious pattern may be matched multiple times along the same logic blocks and signals. Therefore, to avoid repetition, the attacks are consolidated into a list of unique attacks without repetition.

D.2.7.3 Risk Assessment

Risk assessment is the process by which trustability ratings are assigned to each attack. The trustability rating of an attack represents the likelihood that the attack is real as opposed to a false positive, and is formed by taking all known information about the circuit and threat pattern and combining them into an integer number.

The formula for the trustability number calculation is a simple weighted average using the following variables.

- *Threat Pattern Reliability* – This is a number that will be incorporated as part of the Library's self-learning ability. Each malicious asset pattern in the Library contains a reliability number which represents the likelihood that the attack or located malicious asset pattern is real. When self-learning features are implemented in the Library, this number can be updated whenever its malicious pattern is used in an attack. If the attack was actual, the number will be increased, otherwise it will be decreased. Currently, the number represents only a best guess based on the specificity of the corresponding malicious asset pattern.
- *Circuit Application* – When the circuit's intended primary use has been selected, a set of three numbers, or "threat levels", is passed along to risk analysis. Each number corresponds to the likelihood of each type of threat (Denial-of-Service, Data Tampering, and Information Leakage) occurring, given the circuit's application. Each attack, upon creation, is assigned one of the three types of threats, and the Trustability Calculator matches the threat's type with the appropriate number from the circuit's application.
- *Third Party Trustability* – Since Third Party vendors can have different degrees of trustworthiness (some are more trustable than others), each Third Party IP block is "weight". This weight is determined by the user and is the user's best guess as to the likelihood that an IP vendor might have malicious intent.

  After each attack's trustability number has been calculated, all the ratings are

averaged to give the circuit an overall trustability rating. This rating provides a general estimate for the cause for concern in deploying the circuit into the intended application. Once the trustability rating has been calculated, it and all other attack information are sent to the display window.

D.2.8 Results

Upon completion of risk analysis, specific information about the overall circuit design is displayed to the user. A screenshot of the displayed information is shown in Figure 8.

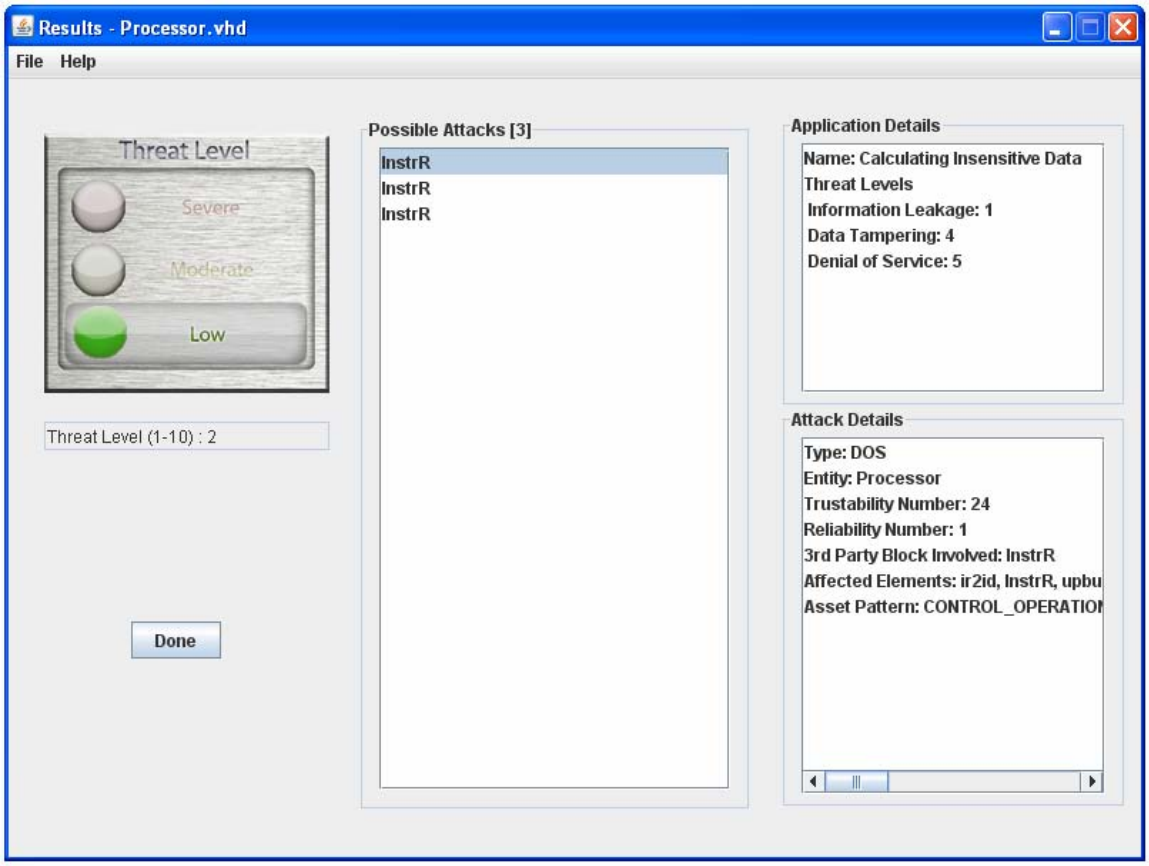

Figure 8: The results after performing risk analysis

There are five important components to the display window.

- **Threat Level Indicator** - The Threat Level Indicator, located in the upper left portion of the display window, offers the overall circuit reliability to the user. It informs the user whether the circuit is considered severe, moderate, or low in terms of threat possibility. It is based on the calculated threat level of the entire circuit, which is located directly below the indicator.
- **Overall Threat Level Number** – The Overall Threat Level Number, located directly below the Threat Level Indicator on the left side of the display, is a numerical representation of the Indicator. It gives a rating to the overall circuit within a range from 1 to 10, with 10 representing the highest threat possibility.
- **List of Potential Attacks** – Shown in the middle of the window is a prioritized

list of potential attacks according to each attack's trustability number, which was determined during risk analysis. The user is able to select any potential attack from the list, and the details of the attack are shown in the "Attack Details" portion of the display located in the lower right corner of the window.

- **Application Details** – In the upper right corner of the display window is information about the application that was selected for the IC. It displays the name of the selected application as well as the threat levels corresponding to the application.
- **Selected Attack Details** – Once an attack has been selected from the list of potential attacks, details about the attack are displayed in the bottom right corner of the display window. The information displayed includes the following:
    - *Type* – the type of threat that the attack corresponds to. This takes on one of three possible values: "DOS" (Denial-of-Service), "DT" (Data Tampering), or "IL" (Information Leakage).
    - *Entity* – the VHDL file the attack occurs in.
    - *Trustability Number* – an integer value, ranging from 1 to 100, representing the likelihood that the attack is real as opposed to a false positive.
    - *Reliability Number* – intended to be a statistical value of the accuracy of the particular attack over time. Due to the nature of this number, it becomes more effective with increasing use of the Tool. It is aimed to answer the question "How many times has this asset pattern been an actual attack in past circuit designs?"
    - *3rd Party IP Block Involved* – the Third Party IP block involved in the attack. Given the organization of the display window, this is always a logic block found inside the VHDL file that the attack occurs in (i.e., the VHDL file corresponding to the *Entity* entry in the display window).
    - *Affected Elements* – the components of the VHDL file that are involved in the attack. These are the components which contained the asset pattern in the Threat Library that was matched when performing risk analysis.
    - *Asset Pattern* – the pattern in the Threat Library that was matched when performing risk analysis

D.2.9 Limitations

At present, the TRUTH Analysis Tool has a limited range of VHDL syntax that it can accurately support when creating the circuit structure S. Modifications are being made in order to expand this range, however, the key obstacles reside in the VHDL parser. Since the circuit structure *S* provides a foundation for the entire TRUTH Analysis Tool (the Tool is only as effective as what the accuracy of the circuit structure *S* allows), modifications have primarily focused on improving the flexibility of the parser. Currently, the parser has the following limitations when parsing a VHDL file:

- *Case Sensitive* – the parser is case sensitive. This means each component needs to have the same name, down to the case of each letter of the name, throughout the

entire VHDL file. For example, suppose a signal was declared with the name being "sig_a". Therefore, throughout the *entire* VHDL file, the signal needs to be addressed as "sig_a" (not "Sig_a", "SIG_A", etc.). The parser considers "sig_a" and "sig_A" to be two different signals, and will handle them as such, causing the Tool to generate an error.

- *Constant Declarations* – the parser does not recognize constant declarations. Any constants that are declared are ignored by the parser.
- *Constants in Component Instantiations (i.e., Port Map statements)* – each component instantiation cannot contain constants. The term "constant", in this context, refers to any value that is not declared as a signal and that never changes value. Two common types of examples include '1' and "0000". Any constants included in component instantiations will cause an inaccurate circuit structure *S* to be created and will generate an error when performing risk analysis.
- *Function Declarations* – the parser does not recognize function declarations. These declarations will not cause the Tool to generate an error but rather are completely ignored by the parser.
- *Generate Statements* – any generate statements found in a VHDL file are ignored by the parser. These statements will not cause the Tool to generate an error but are simply not recognized by the parser.
- *Generic Statements* – the parser does not recognize generic statements. Any generic statements are ignored by the parser which, in turn, will cause an inaccurate circuit structure *S* to be created. For example, if the statement "GENERIC (N: INTEGER := 16);" was found in a VHDL file, the parser would not recognize the meaning behind it. Therefore, whenever 'N' was addressed throughout the file, the parser would only associate it as being a character as opposed to being an INTEGER of size 16.

D.2.10 Testing

Throughout the development of the TRUTH Analysis Tool, several different circuit designs were used for testing and experimentation purposes. At first, since the initial task was to develop the circuit structure *S*, only one design was used for testing. This design, which was a simple RISC microprocessor, was used in verifying that the development of the circuit structure *S* accurately represented its VHDL counterpart. Once this phase was close to completion, several other designs were introduced to the Tool. These included a keyboard controller as well as a 64-bit block cipher, among others. However, even though these designs were used at different times throughout the development stages, the RISC microprocessor served as the primary test case throughout the project's entirety.

When the Tool's development neared completion, two examples were developed to demonstrate the Tool's functionality. Both examples extended the RISC microprocessor design. One successfully performed a Denial-of-Service attack while the other a Data Tampering attack. Upon demonstration, the Tool successfully located both attacks.

D.3 References

[1] S. Smith and J. Di, “Detecting Malicious Logic Through Structural Checking,” 2007 IEEE Region 5 Technical Conference, Apr. 2007

[2] Grimm, Dr. Christoph and J.W. Goethe. Java VHDL Parser Framework. University of Frankfurt, Germany. http://www.ti.informatik.uni-frankfurt.de/grimm/hybrid.html.

[3] S. Myagmar, A. Lee, and W. Yurcik, “Threat Modeling as a Basis for Security Requirements,” Symposium on Requirements Engineering for Information Security (SREIS) in conjunction with 13th IEEE International Requirements Engineering Conference (RE) , Paris, France, August 29th, 2005.